Layout and Content Extraction for PDF Documents

Hui Chao and Jian Fan

Hewlett-Packard Laboratories, 1501 Page Mill Road, ms 1203
Palo Alto, CA, 95129, USA
{hui.chao,jian.fan}@hp.com

Abstract. Portable document format (PDF) is a common output format for electronic documents. Most PDF documents are untagged and do not have basic high-level document logical structural information, which makes the reuse or modification of the documents difficult. We developed techniques that identified logical components on a PDF document page. The outlines, style attributes and the contents of the logical components were extracted and expressed in an XML format. These techniques could facilitate the reuse and modification of the layout and the content of a PDF document page.

1 Introduction

Portable document format (PDF) is a common output format for electronic documents. PDF documents preserve the look and feel of the original documents by describing the low-level structural objects such as a group of characters, lines, curves and images and associated style attributes such as font, color, stroke, fill, and shapes, etc. [1] However, most PDF documents are untagged and don't have the basic high level logical structure information such as words, text lines, paragraphs, logos, and figure illustrations, which makes reusing, editing or modifying the layout or the content of the document difficult. Although originally, the Portable Document Format (PDF) was designed for the final presentation of a document, there were trends to extend the capability of PDF to more than a viewable format [2] and to recover the PDF document layout and its content. In variable data printing (VDP), it is desirable to reuse the layout of an existing PDF page as a template or to reedit the content of an existing page by replacing the images, figure illustrations or modifying a body of text to create a new page or a new version of the page without going back to the original document. The goal of this work is to identify logical components and their associated layouts and to extract the content of the components in a PDF document. The layout and style attributes of the logical components can be used as a template for creating new pages.

In the past two decades, extensive studies have been done in discovering the layout of document images [3-8] for content understanding and extraction. Because of the limited information available in the bitmap images, for example there is no layering information of the page objects, most of the layout studies have been limited to business letters, technical journals and newspapers, where objects overlaps are minimal. Studies have also been done on multilayer analysis and segmentation of document images for the compression purpose. Scanned document images were broken into

S. Marinai and A. Dengel (Eds.): DAS 2004, LNCS 3163, pp. 213–224, 2004.

different regions and different layers based on the texture of the objects on the page [13, 14]. Several groups have studied PDF documents layout and logical structure [9-12]. Brailsford et al. have been able to segment a PDF document page into different image and text blocks. Anewierden has reported his work on recovering the logical structure of the technical manuals in PDF. Hadjar et al. have developed a tool for extracting the structures from PDF documents. However, most of the studies gave limited consideration to overlaid and arbitrarily shaped blocks.

In this paper, we present our techniques that automatically segmented a PDF document page into different logical structure regions such as text blocks, images blocks, vector graphics blocks and compound blocks. Each block represented a logical component of the document. We identified the geometrical outline of each component, extracted the style attributes and content of each component. The results of this analysis were expressed in an XML format.

2 Document Layout and Content Analysis and Extraction

PDF document content stream lists all the page objects such as text objects, image objects, path objects etc. [1] Path objects are referred to as vector graphics objects or drawings and paintings composed by lines, curves and rectangles. Page objects in PDF document don't reflect nor are related to the logical structure or logical components of the document. For example, a text object may only have part of the characters of a word; path objects which are the building blocks for the graphical illustrations, such as bar charts, pie charts and logos are often just a fraction of the whole figure illustrations e.g. one bar in a bar chart [16, 17]. To discover the logical components of a document, we need to analyze and interpret the layouts and attributes of the page objects so as to correctly break or group them into different logical components.

2.1 Overview of the System

Figure 1 shows our processing pipeline. In the documents with high graphical content, there are often blocks overlay, e.g. a text block or a logo is embedded in an image, which makes the analysis more complicated. To minimize the potential logical components overlay and the interference between different logical components, we first separated the page into three layers: text, image and vector graphics layer. Each layer became an individual PDF document. To identify the logical structural component blocks in each layer, we could either directly analyze the three PDF documents or convert the PDF documents into bitmap images and perform document image segmentation. We then obtained the outline of each component block in polygon and extracted the style attributes and content of the component. For the text components, we extracted the font, size, line spacing and color as the text style and extracted the text strings as the content. For the image components, we identified the shapes of the images and the masks if they existed and extracted the data stream of each image into an image file. For the vector graphics components, we identified the path objects within each component, converted the path objects into the SVG path objects and created an SVG file for each vector graphics component.

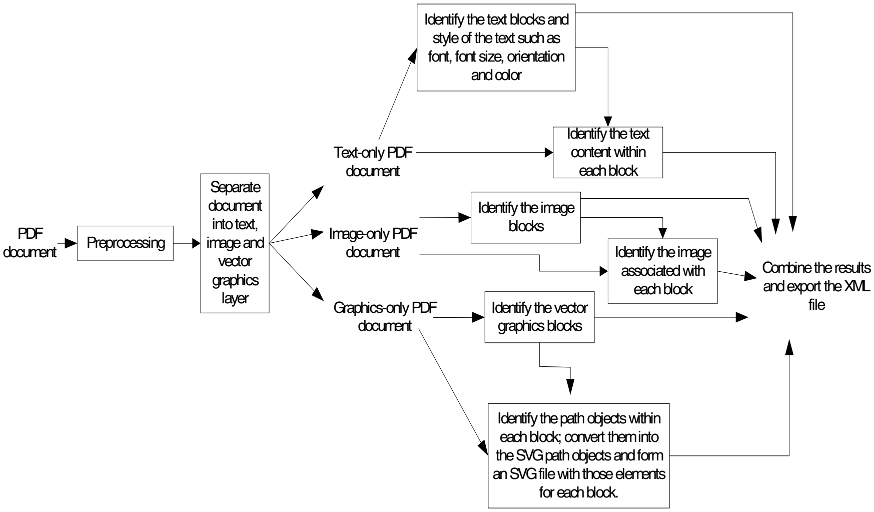

PDF document
Preprocessing
Separate document into text, image and vector graphics layer
Text-only PDF document
Image-only PDF document
Graphics-only PDF document
Identify the text blocks and style of the text such as font, font size, orientation and color
Identify the text content within each block
Identify the image blocks
Identify the image associated with each block
Identify the vector graphics blocks
Identify the path objects within each block; convert them into the SVG path objects and form an SVG file with those elements for each block.
Combine the results and export the XML file

Fig. 1. The overview of the processing pipeline. Each block represents a logical component or subcomponent of the document.

2.2 Preprocessing

Page objects can be simple types like text, image or path objects. They can also be compound types like container objects and form XObjects, which themselves may contain a list of objects including text, image and path objects or even another level of compound objects. In order to access all the objects on the page, objects within compound objects were extracted and replaced in the content stream list to substitute for the compound objects while the layout order of the objects was preserved. The extraction and substitution process continued until there were no more compound objects on the page display list.

2.3 Separating the Document into Text, Image and Vector Graphics Documents

After the compound objects were replaced by the simple-types objects, all the text objects on the page were extracted to form a new text-only PDF document, all the image objects were extracted to form a new image-only PDF document and all the path objects were extracted to form a new path-objects-only PDF document. The order of the objects placed on the page was preserved during this exercise.

2.4 Text Segmentation

For the text on the page, the PDF document provides information of all characters used in the text, their position and attributes, such as font, color, character spacing, orientation, etc. Directly analyzing the PDF document can provide more accurate information than document image analysis and OCR. And the results are not affected by the contrast of the text, the background and text overlay. Therefore, our text segmentation was performed directly on PDF document.

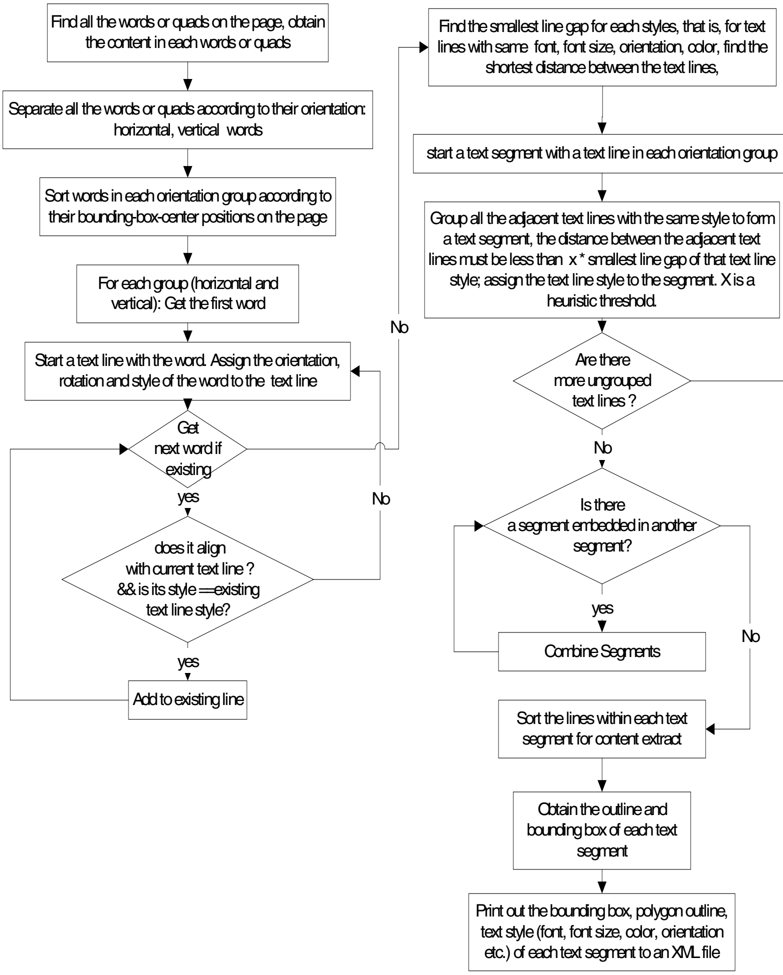

Find all the words or quads on the page, obtain the content in each words or quads
Separate all the words or quads according to their orientation: horizontal, vertical words
Sort words in each orientation group according to their bounding-box-center positions on the page
For each group (horizontal and vertical): Get the first word
Start a text line with the word. Assign the orientation, rotation and style of the word to the text line
Get next word if existing
yes
does it align with current text line ? && is its style ==existing text line style?
yes
Add to existing line
No
No
Find the smallest line gap for each styles, that is, for text lines with same font, font size, orientation, color, find the shortest distance between the text lines,
start a text segment with a text line in each orientation group
Group all the adjacent text lines with the same style to form a text segment, the distance between the adjacent text lines must be less than x * smallest line gap of that text line style; assign the text line style to the segment. X is a heuristic threshold.
Are there more ungrouped text lines ?
Yes
No
Is there a segment embedded in another segment?
yes
Combine Segments
No
Sort the lines within each text segment for content extract
Obtain the outline and bounding box of each text segment
Print out the bounding box, polygon outline, text style (font, font size, color, orientation etc.) of each text segment to an XML file

Fig. 2. The flowchart for text segmentation.

As shown in Figure 2, based on a bottom up approach, text segmentation was started with the smallest text: from characters in PDF documents, words were formed using Adobe's word-finder [20, 21]; from words, text lines were formed based on the alignment, style of the words and distance between them; and finally from text lines, text segments were formed based on text lines adjacency, styles and line gaps.

2.4.1 Forming Text Lines from the Words

PDF documents have the style and position information about characters used in the document. We used an attributes vector, which included font name, font size, color and orientation of the words, to represent the style of the words. Adobe's Acrobat word-finder provides the coordinates of the four corners of the quad(s) of the word. Quad is the quadrilateral bounding of a contiguous piece of a word. Based on the orientation of the words, all the words on the page were separated into two groups, horizontal and vertical words. Assuming the angle between the bottom line of the quad and the horizontal axis of the page was α and $0 \leq \alpha \leq 90^0$, if $0 \leq \alpha \leq 45^0$, the word was treated as horizontal word; otherwise, if $45^0 < \alpha \leq 90^0$, it was treated as a vertical word. Quads in each group were sorted based on the center positions of the quads on the page.

Assuming: (x, y) were the coordinates of the center of the quads and PW and PH were the page width and height. For horizontal words, the sorting was based on the values of x+ y* PW and for vertical words the sorting was based on the values of y + x *PH. Therefore, horizontal words were sorted from left to right first and then from top to bottom and vertical words were sorted from top to bottom first and then from left to right. After sorting, the horizontal words aligned at the bottom were listed next to each other. Similarly, vertical words aligned at the right were listed next to each other.

Text lines were started with the first words of the horizontal and vertical words group respectively. The style of text line adopted the style of the word that started the text line. The next word on the list was selected and compared against the text line. The following criteria were checked: style, alignment and distance between the bounding box of the text line and the bounding box of the new word. The distance between two aligned boxes was the shortest distance between the east and west edge of the two boxes for bottom-aligned boxes and north and south edge of the two boxes for right-aligned boxes. If the word and text line had same style, if they were aligned at the bottom for the horizontal text or aligned at the right for the vertical text and if the distance between the bounding boxes of the word and the text line was less than a predetermined limit, e.g. space character or the maximum font width for the font used in the word and text line, then the word was added to the text line. If the next word on the list didn't satisfy the criteria, another text line was started with this word. The text content within each text line was obtained by concatenating the sorted words in the text line.

2.4.2 Text Line Analysis

For text lines with same style, there is usually a consistent line gap between text lines that belong to the same text segment or same paragraph. A larger gap is usually present between two text segments. Text line analysis was performed to search for the consistent line gap for each text line style on the page. The values of the line gaps were used later in forming the text segments.

After all the text lines were formed, the distances between any two adjacent text lines with the same style and orientation were measured. The adjacency referred to two text lines with no objects sitting in the gap between them. The value of the text line gap for a particular text style was the shortest vertical distance for the horizontal text and the shortest horizontal distance for the vertical text. The distances were measured between the bottoms of the current text line to the top of the next line.

2.4.3 Forming Text Segments from the Text Lines

Text segments were formed by grouping adjacent text lines within close distance, with same style and orientation. A text segment was started with a text line. Using this text line as reference, all text lines that were adjacent or attached to this text line, had the same style as this text line and were with the distance shorter than a threshold to this text line were added to the text segment. The threshold was a function of the line gap and the font size for the particular text line style. We chose the threshold to be the minimal of the twice of the line gap and twice the height of the bounding box of the reference text line. The text lines that were added to the text segments were marked as the grouped text lines. Using the newly added text lines as references, with same criteria, more adjacent or attached lines were added to the text segment. This process was repeated until there were no more ungrouped text lines that could be added to the current text segment. Each text line could only belong to one text segment. The next segment started with another ungrouped text line. The same process was performed to form all the text segments on the page.

2.4.4 Combining the Text Segments

Sometime, one or more words within a paragraph were highlighted by having different style such as italic while the majority of the words were regular; superscriptions or subscriptions might be present with different font size and different alignment from the main text. Previous steps performed for the text segmentation would identify them as individual text segments. Since our goal was to find the design intention for the logical blocks, which should be minimally affected by the current content, in this step, we combined the small text segments with the big one if the small text segments were embedded in the big one. The attributes or the style of the combined segment were the attributes or the style of the big text segment, which represented the style of the majority of text within the block.

2.4.5 Tracing the Outlines of the Text Segments

In the graphical rich documents, we often saw a text block wrapping around another arbitrarily shaped image or graphics block. It was important to find the outline of the text block so that the shape and the layout relation between blocks would be preserved during text content replacement or modification. As shown in Figure 4, the outline of a text segment was identified by tracing the corner points of the bounding boxes of the sorted text lines within the segment. Text lines within a horizontal text segment were sorted from the left to the right first and then the top to the bottom. Text lines within a vertical text segment were sorted from the top to the bottom first and then the left to the right. To find the outline of a horizontal text segment, as shown in figure 4a, we first connected all the left corner points of the bounding boxes of the text lines within the segment till the last text line; then moved to the right bottom corner of the bounding box of the last text line; and connected all the right corner points. When connecting points on the left side, only a move down was allowed and when connecting points on the right side, only a move up was allowed. Processing steps were as follows:

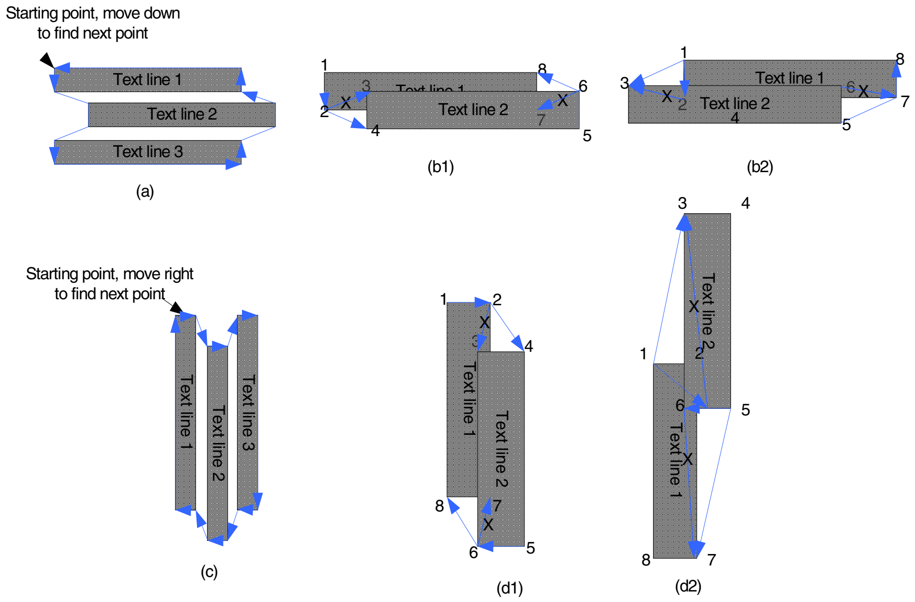


Fig. 3. Tracing the outline of text segment by connecting the corner points of the bounding boxes of text lines. (a) For horizontal text lines, connections started with the text line on the top, moved down to connect all the left side points, then moved up to connect all the right side points of the bounding boxes of the text lines. (b1) When connecting the left side points, only moving down was allowed. (b2) when connecting the right side points, only moving up was allowed. (c)For vertical text lines, connection started with the text line on the most left, moved right to connect all the points on the top and then moved left to connect all the points at the bottom of the bounding boxes of the text lines. (d1) when connecting the points on the top, only moving right was allowed. (d2) when connecting the points at the bottom only moving left was allowed.

```
Assuming origin is at the lower left corner,
1. Sort all boxes by the vertical positions of the centers from the top to bottom $B_0, B_1, \ldots B_{n-1}$
2. Create two point-vectors $L$ and $R$ for the left and right points
3. Fill L with the left top and bottom points of the bound boxes of the text lines, starting with
   the bounding box on the top: $[L_0, L_1, \ldots L_{2n}]=[B_0.L.top, B_0.L.bottom, B_1.L.top, B_1.L.bottom, \ldots B_{n-1}.L.top, B_{n-1}.L.bottom]$ ;
4. Fill R with the right bottom and top points of the bound boxes of the text lines, starting with
   the bounding box at the bottom: $[R_0, R_1, \ldots R_{2n}]=[B_{n-1}.R.bottom, B_{n-1}.R.top, B_{n-2}.R.bottom, B_{n-2}.R.top, \ldots B_0.R.bottom, B_0.R.top]$ ;
5. Set $NumPtsL=2*n; NumPtsR=2*n; Count=0;$
6. do{    count=0;
          for I=0:NumPtsL
          if connecting $L_i$ to $L_{i+1}$ is a right up move (i.e. $L_i.x< L_{i+1}.x$ and $L_i.y< L_{i+1}.y$),
                  Remove $L_{i+1}$ from L[], count++; NumPtsL--;( figure 4(b1))
          else if connecting $L_i$ to $L_{i+1}$ is a left up move (i.e. $L_i.x> L_{i+1}.x$ and $L_i.y< L_{i+1}.y$)
                  Remove $L_i$ from L[], count++; NumPtsL--;( figure 4(b2))
   }while(count>0)
```

```
7. do{   count=0;
         for I=0:NumPts-1
         if connecting $R_i$ ->$R_{i+1}$ is a right down move (i.e. $R_i.x< R_{i+1}.x$ and $R_i.y< R_{i+1}.y$),
                 Remove $R_i$ from R[], count++; NumPtsR--; ( figure 4(b2))
         else if connecting $R_i$ ->$R_{i+1}$ is a left down move (i.e. $R_i.x> R_{i+1}.x$ and $R_i.y< R_{i+1}.y$)
         Remove $R_{i+1}$ from R[], count++; NumPtsR--;( figure 4(b1))
     }while(count>0)
```

8. Connect all the points in the order of L_0 -> L_2, …->$L_{NumPtsL-1}$ $\rightarrow$ R_0 ->R_2, …->$R_{NumPts-1}$ -> L_0 where *NumPtsL* is the number of points in the set *L* and *NumPtsL* is the number of points in the set *R*.

Similar to horizontal lines, to find the outline of a vertical segment, we first connected all the top corner points of the bounding boxes of the sorted text lines, then connected all the bottom points of the bounding boxes of sorted text lines as shown in figure 4c. When connecting points on the top, only a move right was allowed; when connecting points at the bottom, only a move left was allowed as shown in figure 4d. The outlines of the segments were expressed as the polygons. The connecting points were the vertex of the polygons.

2.4.6 Extracting the Content of the Text Segments

After all the text segments were identified, text contents within the text segments were formed by concatenating the text strings of the sorted text lines within the segments.

2.5 Image Components Identification and Extraction

To identify the image components on the page, we could convert the image-only PDF document to a bitmap image and apply document image segmentation. However, since image objects in PDF are well-defined logical units (different from text and path objects). Image components were identified directly from PDF document.

Each image object in the PDF document page content stream was identified as a logical component of the document, the image component. The image data stream was extracted and saved as an image file. If the image had a mask that gave the special effect to the image or a clipping path that defined the visible portion of the image, the mask and clipping path were extracted and the image object was saved as an SVG image object and an SVG file would be created. An XML file, which described the bounding box of the image component on the page, the width and height of the image and the type of the image decoder, was created. It also referenced the image file name or the SVG file name.

2.6 Vector Graphics Segmentation and Extraction

To find the vector graphics component blocks on the page, we had the choice of analyzing PDF directly [16] or applying document image segmentation technique in the automatic mode [18, 19]. We found in a lot of cases, for vector graphics components, it was more straightforward to analyze the bitmap of the page than to follow the drawings of the path objects on the page. Therefore we applied document image segmentation tool to find the regions that represented different components. The outline of each region was used to find the path objects within each component on the PDF page. With those path objects, we created the SVG representation for the logical component.

2.6.1 Vector Graphics Zoning

We converted the path-objects-only PDF page into a bitmap image and applied a document image segmentation tool to the bitmap image. Different zones were identified. Each zone represented a logical component. The outline of each zone was expressed in a polygon.

2.6.2 Identifying the Path Objects Within Each Block

In PDF document, drawings or path objects are expressed as a combination of lines, curves and rectangles. In this step, the coordinates of the polygon outline of each region in the bitmap image coordinate space were converted to the coordinates in PDF page coordinate space, the path objects enclosed in the polygons were identified and the corresponding SVG objects were created.

To find out if a path object in the PDF document belonged a region, we could check if all the points along the lines, curves and rectangles of the path objects were within the polygon outline of the region. To reduce the computing complexity, we only checked the starting and the ending points of the lines and curves and the four corner points of the rectangles. If and only if the starting and end points for lines and curves and four corner points of the rectangles of all the sub-segments of a path object were inside the polygon [22], then the path object was identified as part of the logical component. All the identified path objects were translated into SVG path objects. An SVG file was created for each logical component.

If there were still unidentified path objects left, which didn't belong to any existing logical components, the unidentified path objects were grouped to form new logical components based on their attachment, vicinity and layering order [17]. Again an SVG file was created for each logical component.

2.7 Combining Components into a Compound Component

If a component block discovered in 2.4-2.6 was completely embedded in another block, the two components were most likely to be one logical component. Therefore in this step, they were combined into a compound component with two subcomponents. This process was repeated until there was no one component being completely embedded in another component. The subcomponents remained to be accessible logical units that could be modified or replaced.

3 Results

Figure 4 shows the results of applying our tool to a PDF page at different processing steps. The outlines of different segments were highlighted, with the text segments in green, the image segments in red and the vector graphics segments in blue. The text contents were expressed in XML files. Images were saved as image or SVG files and vector graphics components were saved as SVG files.

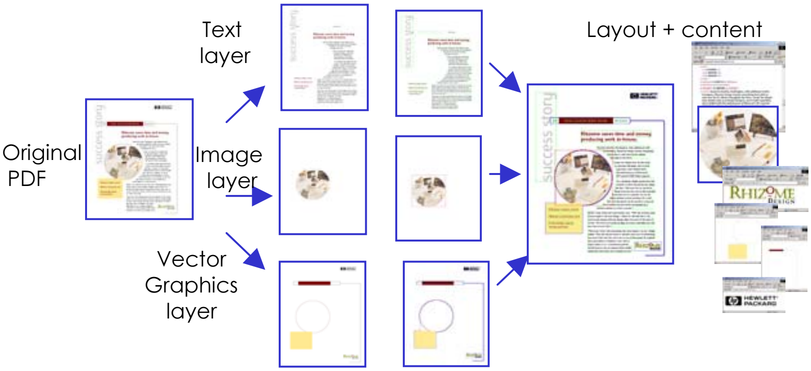
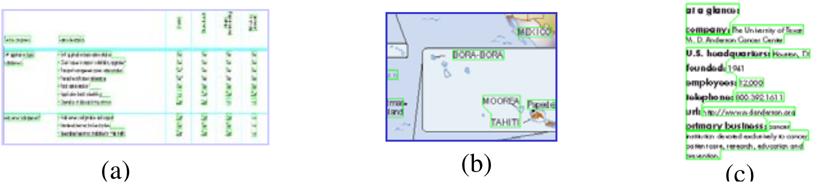


Fig. 4. An example that shows how the system works by giving the results at each processing step. The page was separated into text, image and vector graphics layer; each layer was analyzed to identify logical-component blocks. The style and content of the components were extracted.

We have tested our tool on 100 pages of HP marketing collaterals and 100 pages of traveling brochures. Segmentation results were manually checked. Segmentation errors happened in 18 pages among these 200 pages. Errors were in three categories: (1) pages with tables, an example is shown in figure 5a, in which the text in different cells were grouped into the same segment; (2) text on a map, an example is shown in figure 5b, in which text that were close to each other, but indicate different locations on the map were grouped into the same segment; (3) text style changing after a colon, an example is shown in figure 5c. To overcome these errors, methods to identify special kinds of logical components need to be developed, such as table and map recognition. A user instruction would also help to improve the precision of results.


MEXICO
BORA-BORA
MOOREA
TAHITI
at a glance:
company: The University of Texas M. D. Anderson Cancer Center
U.S. headquarters: Houston, TX
founded: 1941
employees: 12,000
telephone: 800.392.1611
url: http://www.mdanderson.org
primary business: cancer institution devoted exclusively to cancer patient care, research, education and prevention.

(a) (b) (c)

Fig. 5. Examples show the segmentation errors. The segmentation error happened when the text were embedded in a table (a); when text were embedded in a map (b); and when the text style changed after the colons (c).

4 Conclusion and Discussion

We have developed techniques to identify logical components on a PDF page. The outlines of the segments were determined and the style and content within the components were extracted and expressed in an XML file. A user could treat the outline and style attributes of the logical components on the page as a layout template for the new page creation. A user could also take a logical component to modify it or to reuse it in another document.

However, page layout is more than the outline of the document, it is also the relation constraints between the logical components such as alignments and white spaces. Further research need to be done to discover the layout constraints so that a more dynamic layout reuse can be allowed.

Acknowledgement

We want to thank our colleague Steve Simske for providing the document image segmentation tool, Lei He at Atlanta Armstrong University for testing the image segmentation tool and developing user interface for our tool.

References

1. “PDF Reference”, Adobe system Incorporated.
2. “Acrobat 4: Adobe’s bid to make it more than a viewer” Walter, M. Seybold report on Internet Publishing. Vol. 3, no. 7.
3. Nagy G., Seth S. and Viswanathan M., “A prototype document image analysis system for technical Journals”, IEEE Computer, vol. 25, pp. 10-22, 1992.
4. Dengel, A. “Initial learning of document structures”, the Second International Conference on Document Analysis and Recognition (ICDAR),1993 pg 86-90.
5. Dengel, A., Dubiel, F.“Clustering and classification of document structure? A machine learning approach.” ICDAR 1995, pp 587-591.
6. Kise, K. Incremental acquisition of knowledge about layout structures from examples of documents. ICDAR 1993, (pp.668-671).
7. Walischewski, H., Automatic knowledge acquisition for spatial document interpretation.” Proceedings of the Fourth ICDAR 1997, (pp. 243-247).
8. Jain A., Yu, B., “Document representation and its application to page decomposition”. IEEE Transactions on pattern analysis and machine intelligence, Vol. 20, No. 3, March 1998.
9. Smith, P.N., Brailsford, D.F., “Towards structured, block-based PDF”, Electronic publishing: origination, dissemination and Design, Vol. 8 no. 2-3.
10. Lovegrove W.S., Brailsford, D.F “Document analysis of PDF files; method, results and implications”, Electronic publishing, Vol. 8 no. 2-3.
11. Anjewierden A.,“AIDAS: Incremental logical structure discovery in PDF document“, ICDAR 2001.
12. Karim, H., Rigamonti, M.,Lalanne, D., Ingold, R. “Xed: a new tool for extracting hidden structure from electronic documents”. Digital Library workshop. Palo alto, January 2004.
13. Haffner, P., Bottou, L., Howard, P., Cun, Y. L. “DjVu Analyzing and Compressing Scanned Documents for Internet Distribution”. ICDAR, pp 625-628, 1999.
14. Jian Fan, "Text extraction via an edge-bounded averaging and a parametric character model", Proc. Electronic Imaging 2003, SPIE Vol. 5010, pp. 8-19, 22-24 Jan. 2003, Santa Clara, CA.

15. Chao, H, Beretta, G, Sang, H, “PDF Document Layout Study with Page Elements and Bounding Boxes”, DLIA 2001.
16. Chao, H., “Graphics Extraction in PDF Document”, SPIE vol. 5010, no. 317.
17. Chao, H., “Background extraction in multi-page PDF document”, DLIA 2003.
18. Simske, S.J., Arnabat J., “User-directed analysis of scanned images”, ACM workshop of Document Engineering 2003.
19. Revankar, S.V. and Fan, Z., “Image Segmentation system”, U.S. Patent 5,767,978, January 21, 1997.
20. Paknad, M.D. and Ayers R M., “Method and apparatus for identifying words described in a portable electronic document”, US patent 5,832,530.
21. Ayers; Robert M., “Method and apparatus for identifying words described in a page description language file”, US patent 5,832,531.
22. O’Rourke, “Computational Geometry in C”, 1998.